VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

Katedra informatiky

Mapa hvězdné oblohy
Skymap

2014 Jiří Kupka

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Jiří Kupka**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Mapa hvězdné oblohy
Skymap

Zásady pro vypracování:

Cílem práce je navrhnout a naimplementovat aplikaci pro zobrazení hvězdné oblohy v prostředí webového prohlížeče.

Ve své práci uveďte:
1. Nastudujte promítaní nutná pro zobrazení hvězdné oblohy a řádně je popište.
2. Naimplementujte zobrazení hvězdné oblohy alespoň s hvězdami známých souhvězdí, planet sluneční soustavy a Měsíce,
3. Implementaci proveďte s využitím HTML5 technologií.
4. Aplikace bude reagovat na uživatelské vstupy a bude doprovázena databází, kterou bude moci uživatel přehledně procházet.
5. Svou aplikaci řádně otestujte a v textu zhodnoťte.

Seznam doporučené odborné literatury:

[1] Wolf, M.: Astronomická příručka. Vydání 1., Praha, nakladatelství Československé akademie věd, 1992. ISBN 80-200-0467-X
[2] Široký, J., Široká, M.: Základy astronomie v příkladech. Vydání 2., Praha, Státní pedagogické nakladatelství, 1973
[3] Karkoschka, E.: Astronomický atlas hvězdné oblohy. Vydání 1., Ostrava, 1995. ISBN 80-85606-67-4

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Gaura**

Datum zadání: 01.09.2013
Datum odevzdání: 07.05.2014

VYSOKÁ ŠKOLA BÁŇSKÁ - TECHNICKÁ UNIVERZITA OSTRAVA
Fakulta elektrotechniky a informatiky
1

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26 *Studijního a zkušebního řádu pro studium v bakalářských programech* VŠB-TU Ostrava.

V Ostravě dne 7. května 2014 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 7. května 2014 .....

Chtěl bych poděkovat Ing. Janu Gaurovi za ochotu ujmout se vedení této bakalářské práce a za čas, který mi při její vypracování věnoval.

Abstrakt

Cílem práce je vytvořit mapu hvězdné oblohy za pomocí moderních technologií HTML5 a jazyku Dart v prostředí webového prohlížeče. Mapa zobrazuje nejjasnější hvězdy – ty, které jsou teoreticky viditelné i pouhým okem, planety Sluneční soustavy, Slunce a Měsíc. Aplikace reaguje na uživatelské vstupy, po kliknutí na vykreslený objekt zobrazuje podrobné informace o jeho poloze, existuje možnost jejich vyhledání. Uživatel může mapou volně pohybovat, nastavovat svou pozici a přesunout se v čase do okamžiku, který ho zajímá.

Cílem práce je také prozkoumání možností jazyka Dart, který by měl nabízet dostatečně vhodné a výkonné prostředí pro zpracování a vykreslení velkého množství dat. Součástí práce je také porovnání výkonu ve webových prohlížečích a jazycích JavaScript a Dart.

Klíčová slova: Skymap, Dart, JavaScript, HTML5, MVC, sférická astronomie, souřadnicové systémy, čas, promítání.

Abstract

The goal of the bachelor thesis is to create sky map using modern technology like HTML5 and Dart language in web browser environment. The sky map shows the brightest stars – stars that are visible with the naked eye, planets of the Solar system, the Sun and the Moon. The application responds to the userś inputs, shows detail information about position of the clicked object and user can also search the database of objects. User can move with the map, set his position and jump into time moment, that is interesting for him.

The goal of the thesis is also to explore capabilities of the Dart language, which should provide suitable and efficient environment for processing and rendering of large amounts of data. Part of the thesis is also performance comparison in web browsers and JavaScript and Dart language.

Keywords: Skymap, Dart, JavaScript, HTML5, MVC, spherical astronomy, coordinate systems, time, projections.

Seznam použitých zkratek a symbolů

AJAX	– Asynchronous JavaScript and XML
API	– Application Programming Interface
FPS	– Frames Per Second
FTP	– File Transfer Protocol
GPS	– Global Positioning System
HTML	– HyperText Markup Language
NASA	– National Aeronautics and Space Administration
RIA	– Rich Internet application
SDK	– Software development kit
UTC	– Coordinated Universal Time
WebGL	– Web Graphics Library
XML	– Extensible Markup Language
YAML	– YAML Ain't Markup Language; formát pro serializaci dat čitelný člověkem

Obsah

Seznam tabulek

Seznam obrázků

Seznam zdrojových kódů

1 Úvod

Již od pradávna lidstvo vzhlíželo ke hvězdám. Pravděpodobně již první lidé sledovali noční oblohu s nekonečným obdivem a božskou pokorou. Toto prosté pozorování se časem díky lidské zvědavosti a potřebám prohlubovalo v poznání a pochopení. Astronomie jako věda je jednou z věd nejstarších a její počátky sahají někde k 3-4. tisíciletí před naším letopočtem. Tehdy šla ruku v ruce s vývojem zemědělství, snahou upřesnit počítání času nebo zjištění polohy člověka na Zemi a následné navigace.

Lidé si všimli, že se hvězdy na obloze během noci pohybují po určité dráze, která jak dny v roce plynou, se mírně mění. Tento objev vedl ke snaze pohyb zachytit, zaznamenat a využít k potřebám člověka. V době antiky se objevují první konkrétní jména hvězdářů - astronomů. Z nich je pravděpodobně nejslavnější **Hipparchos**, který vyvinul trigonometrii, vymyslel nové přístroje pro měření polohy, určil délku roku s přesností na 6 minut nebo sestavil katalog s více než 850 hvězd. Pravděpodobně byl také vynálezcem **astrolábu**, který se dlouhou dobu používal jak pro navigaci, tak zjištění aktuálního času podle zeměpisné délky a dalším. Jeho jméno jako poctu nese například kráter na Měsíci, Marsu nebo dokonce satelit NASA (jedná se o akronym **High precision parallax collecting satellite** – **Hipparcos**) a katalog hvězd, ze kterého čerpá data i tato práce. Astroláb byl postupně v různých odvětvích nahrazen za přístroje jiné – například v mořeplavectví za sextant, který byl jednodušší a praktičtější.

Jisté pokusy o zakreslení hvězd jsou známy už od našich předků v pravěku, kdy pravěký člověk výrazné *světélka na obloze* zakresloval do hlíny nebo tesal do kamene. Byla to snaha jistě hodna jejich tehdejší bystrosti, avšak bez souřadnicových systémů se nedalo hovořit o mapě hvězdné oblohy. Ve starověku a středověku byla obloha sídlem bohů a bájných tvorů, které astronomové a umělci viděli ve skupinách jasných hvězd. Na mapách byly tyto postavy zakresleny rukou umělců a jejich krásu obdivovali i obyčejní lidé. Souhvězdí však nepokrývaly celou oblohu a některé hvězdy prostě do žádného z nich nepatřily. Až v roce 1925 se ustanovilo 88 souhvězdí a jejich přesné hranice. Všechny hvězdy od té doby patří do jednoho z těchto zvolených souhvězdí. Nevýhoda zakreslených map je ta, že jsou aktuální pouze v době zakreslení - hvězdy se od sebe vzdalují, planety se pohybují; čas při svém nekonečném běhu všechno mění. Proto se k mapě vždy přikládá údaj, pro kterou **epochu** je mapa platná.

Jak jde pokrok neustále kupředu, vznikají nové a nové mapy hvězdné oblohy za pomocí nejrůznějších technik a médií. Nejpřesnější polohu hvězd nám v dnešní době dává výše zmíněná družice **Hipparchos** jejíž výstupem jsou katalogy hvězd, které následně slouží jako vstupní data aplikací nahrazující *klasické* mapy hvězd. Díky vývoji počítačů a elektroniky se nám do rukou dostávají nástroje, pomocí nichž si snadno můžeme zobrazit hvězdy, které se nám právě nacházejí nad hlavou nebo si zobrazit jejich následnou pouť po noční obloze. Naskytá se nám příležitost připravit se tímto na různé zajímavé astronomické jevy - západ Slunce, Měsíce nebo přelety komet. Nelimituje nás čas, místo, ani stav počasí, které trápilo astronomy minulých století. Nic nebrání člověku od poznávání vesmíru.

2 Souřadnicové systémy a práce s časem

2.1 Sférická astronomie

Sférická astronomie je část astronomie, která definuje nebeskou sféru jako místo, kde se promítají vesmírné objekty. Ve skutečnosti tělesa leží různě daleko od pozorovatele, avšak při takto velkých vzdálenostech můžeme jejich rozdíly zanedbat a pro tento účel říci, že všechna tělesa leží v nekonečnu. Sférická astronomie zavádí některé důležité pojmy a systémy souřadnic, kterými se tato kapitola zabývá.

Základní rovina

Nejčastěji uvažujeme dvě základní roviny. První z nich je rovina ekliptiky, druhá je rovina rovníku. **Rovina ekliptiky** je tvořena pohybem Země kolem Slunce. S časem se téměř nemění, používá se proto jako počátek ekliptikálních souřadnic a uzavírá se světovým rovníkem (rovinou rovníku) úhel přibližně $23°27'$. Souvisí s **póly ekliptiky**, což jsou body na nebeské sféře, které jsou k ní kolmé a procházejí středem Země. Póly ekliptiky svírají s těmi zeměpisnými/světovými již výše zmíněný úhel $23°27'$ (sklon rotační osy).

Průsečíky roviny ekliptiky s rovinou rovníku se nazývají **jarní a podzimní bod**. Den, kdy se Slunce nachází v těchto bodech je označován jako *jarní a podzimní rovnodennost*. Jarní bod se často používá jako počátek astronomických souřadnic. Svůj počátek zde mají například rovníkové souřadnice II. druhu (kapitola 2.3.2).

Pozorovatel

Jedním z důležitých systému souřadnic jsou souřadnice horizontální (kapitola 2.3.3). Ty zavádějí kromě azimutu a výšky také pojmy **astronomický horizont**, **zenit** a **nadir**. Astronomický horizont je rovina, která se odvíjí od polohy pozorovatele; protíná oko pozorovatele a nebeskou sféru. Zenit neboli nadhlavník je pomyslný bod na nebeské sféře ležící přímo nad pozorovatelem. Nadir čili podnožník je naopak bod ležící na nebeské sféře přímo pod pozorovatelem.

Vzdálenost

Sférická astronomie sice předpokládá, že promítané objekty leží v nekonečnu, avšak nejen pro výpočty souřadnic je důležité znát, jak jsou daleko. Jelikož se jedná o skutečně velké vzdálenosti, jednotky používané pro běžné měření na Zemi, jako jsou metry a kilometry, se zdají být v mnoha případech nevyhovující. Zavedli jsme proto jednotky nové, jako například **astronomická jednotka** nebo **světelný rok**. Astronomická jednotka je původně definována jako vzdálenost Země od Slunce. Tato vzdálenost je však proměnlivá, proto v roce 2012 byla přesně definována jako 149 597 870 700 m s jednotkou **au** (astronomical unit). Světelný rok je vzdálenost, kterou urazí světlo ve vakuu za jeden juliánský rok (kapitola 2.2.4) s jednotka **ly** (light year). Velikost jednoho světelného roku je 9 460 730 472 580 800 m.

2.2 Vesmír a čas

Čas je relativní, jak řekl Einstein. V mnoha ohledech se jedná o lidský vynález, který se odvíjí od našeho chápání. Člověk vnímá čas jako lineární veličinu, která putuje neustále a pouze vpřed. V dnešní době říkáme, že čas je veličina závislá na prostoru a hovoříme tak o časoprostoru. Je toho mnoho, co o času ještě nevíme, dlouho nedokážeme pochopit nebo nikdy chápat nebudeme; je však jasné, že existuje, plyne a týká se nás všech.

Člověk chodí do práce, kde musí být přesně *na čas*. Má schůzky, chodí spát v určitou *denní dobu* a do důchodu půjde *za 45 let*. Abychom mohli pracovat s časem a alespoň si ho našemu chápáním přiblížit, stanovili jsme si jednotná pravidla a jednotky. Lidé dávno věděli, že se střídá den s nocí, ale chtěli toto období nějak přesněji zachytit. Z počátku byl den rozdělen na roční období, den a noc. Den jako takový bylo však vhodné rozdělit na menší časové úseky, a tak vznikly hodiny, minuty a později drobnější a drobnější jednotky času. Čas se měřil kyvadly, které později nahradily hodinové strojky a v moderní době atomové hodiny, pomocí kterých získáme nejpřesnější čas, a které slouží jako referenční časomíra hodinám dalším.

V běžné řeči a každodenním životě se setkáme se **světovým** a **pásmovým časem**. Pro potřeby astronomie je však vhodnější počítat s jiným způsobem měření - například **hvězdným časem** nebo **juliánským datem**.

2.2.1 Světový a pásmový čas

Světový čas je nejpodstatnější pro běžné užití. Je to místní čas **nultého (greenwichského) poledníku**. Od něj se postupně odvíjel například čas UTC (Koordinovaný světový čas), který se řídí atomovými hodinami a dnes hojně používaný.

Pásmový čas dělí poledníky po 15°, což nám dává celkem 24 časových pásem. Směrem na východ se pro každé následující časové pásmo přičítá k UTC 1 hodina. Například pro střední Evropu je to UTC+1:00. Bohužel, střední Evropa má ještě specifikum ve střídání letního a zimního času, proto je v zimním období UTC+1:00 a v letním UTC+2:00.

2.2.2 Atomový čas

Atomové hodiny jsou zatím nejpřesnější počítadlo času. 1 sekunda je určena 9 192 631 770 kmity izotopu **cesia 133**. Přesnost atomových hodin se čím dál více zpřesňuje a nyní je na odchylce 1 sekundy za 23 miliard let [10]. Atomové hodiny jsou také důležitou komponentou v navigaci a systému GPS, kde se měří doba putování signálu vyslaného družicí uživateli.

2.2.3 Hvězdný čas

Neboli **siderický čas** je definován jako hodinový úhel jarního bodu. Jeden den hvězdného času trvá 23 hodin, 56 minut a 4,09 sekund. Rozdíl oproti slunečnímu dnu (24 hodin) je způsoben tím, že Země kromě pohybu kolem své osy vykoná ještě pohyb na dráze kolem Slunce. Za pomoci hvězdného času můžeme z rovníkových souřadnic objektů získat jejich obzorníkové (horizontální) souřadnice a tudíž jeho aktuální pozici na obloze.

2.2.4 Juliánské datum

Juliánské datum (JD) zavedl na počátku 17. století francouzský matematik a astronom **Joseph Sceliger**. Jedná se o počet dní od poledne 1.1. 4713 před Kristem. Hodiny a minuty se vyjadřují jako část dne za desetinnou čárkou. Toto datování se velmi často používá v astronomii, jelikož nemusí brát ohledy na přestupné roky nebo počet dní v měsíci. Často se také setkáme s **modifikovaným juliánským datem**, která je zmenšené oproti JD o 2400000,5 dne. Začíná tak 17.11.1858 0:0. [1, str. 61]

Zdrojový kód 1: Algoritmus pro výpočet Juliánského data

```
var year, month, day;
...
if(month<=2) {
    year -= 1;
    month += 12;
}

A = 2-floor(year/100)+floor((year/100)/4)
JD = floor(365.25(year+4716))+floor(30.6001(month+1))+day+A-1524.5;
```

J2000.0 je aktuální epocha Juliánského data. Je to hodnota 2451540.5 nebo taky 1.1.2000 11:58:55:816 UTC.

2.3 Souřadnicové systémy

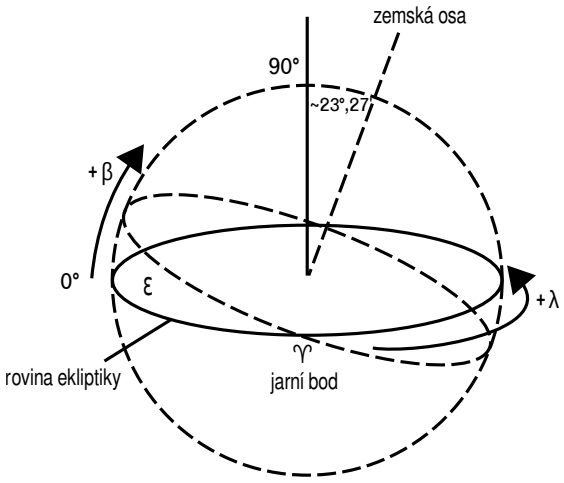
Pokud chceme určit polohu nebeského objektu v prostoru, musíme si definovat vztažnou soustavu a v ní rozumné souřadnice a jednotky. Souřadnicovým systémem říkáme, odkud se na objekt budeme dívat (střed souřadnic) a jakým směrem povedou koordináty. Mezi soustavami je možné souřadnice transformovat. V bakalářské práci se zaobírám pouze některými z existujících souřadnic a souřadnicových systémů. Kromě níže vypsaných existuje spousta dalších, vyhovujících pro konkrétní účel implementace.

Základní rozdělení souřadnicových systémů můžeme definovat podle toho, kde má systém počátek souřadnic. Používáme především **geocentrické souřadnice**, které mají počátek ve středu Země, **heliocentrické souřadnice** se svým počátkem ve středu Slunce a **topocentrické souřadnice** s počátkem souřadnic v místě pozorovatele.

2.3.1 Ekliptikální souřadnice

Souřadnicový systém často používaný pro popis polohy planet Sluneční soustavy. Jedná se o pravotočivý systém, kde základní rovinou je **ekliptika** a počátek číslování je v **jarním bodě**. Zápis souřadnic může být jak ve sférické, tak kartézské podobě.

- **Šířka** β: Úhel $0 - 360°$ s počátkem v jarním bodě. Hodnota roste směrem na východ.
- **Délka** λ: Úhel $-90 - 90°$, kde rovina ekliptiky je $0°$ a směrem k severnímu pólu ekliptiky hodnota roste a jižnímu pólu ekliptiky klesá.

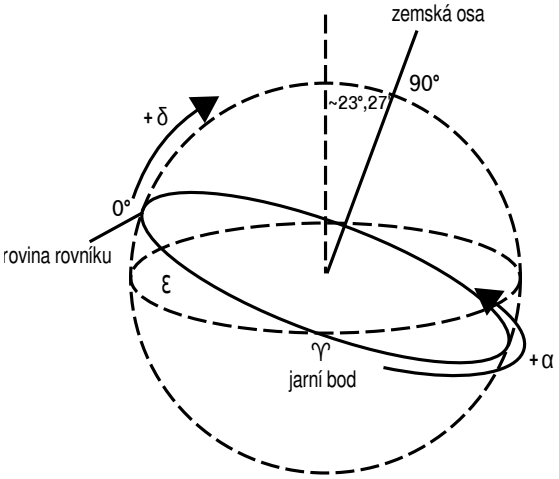


Obrázek 1: Ekliptikální souřadnice

2.3.2 Rovníkové souřadnice II. druhu

Souřadnicový systém často používaný pro zápis polohy hvězd. Jako základní rovina je použita rovina rovníku. Opět se jedná o pravotočivý systém. Rovníkové souřadnice jsou dvojího druhu, kde souřadnice I. druhu používají jako počátek číslování průsečík **meridiánu** a roviny rovníku. Souřadnice II. druhu používají **jarní bod**.

- **Rektascenze (RA)** α: Udává se buď v úhlech $0 - 360°$ nebo v hodinách $0 - 24$. Začíná se v jarním bodě a hodnota roste směrem na východ.
- **Deklinace (DE)** δ: Udává se ve stupních $-90 - 90°$, kde rovina rovníku je $0°$ a směrem k severnímu světovému pólu hodnota roste a jižnímu světovému pólu hodnota klesá.

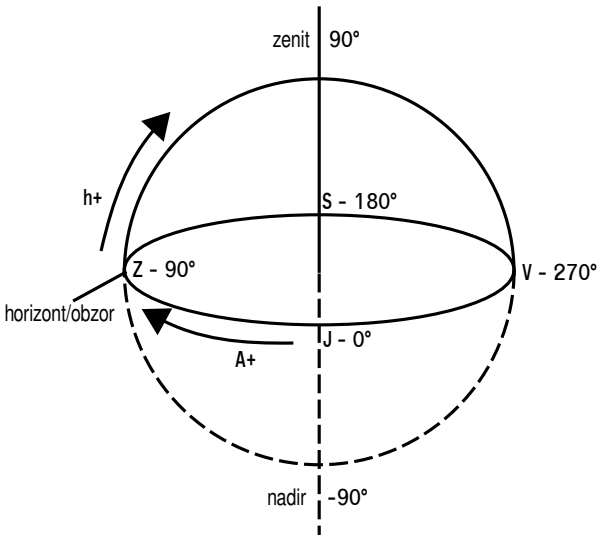


Obrázek 2: Rovníkové souřadnice

2.3.3 Horizontální souřadnice

Horizontální souřadnice jsou pravděpodobně pro běžného pozorovatele nejzajímavější. Základní rovinu tvoří astronomický horizont, který rozděluje nebeskou sféru na dvě poloviny, kde jedna (horní) je viditelná pozorovatelem a druhá (dolní) leží *pod horizontem*, kde Země brání pozorovateli objekty vidět. Počátek číslování je na **severu**. Polárka má například hodnotu azimutu vždy blízkou 0°. Horizontální souřadnice se mění s místem pozorování a časem. Souřadnice pozorovaných těles se z pohledu pozorovatele mění neustále v průběhu dne.

- **Azimut (A)**: Úhel $0-360°$ s počátkem na severu. Hodnota roste směrem na východ. (sever = 0°, východ = 90°, jih = 180° a západ = 270°) [1, str.91]
- **Výška (h)**: Úhel $-90-90°$ s počátkem v rovině horizontu. Hodnoty rostou směrem k zenitu (90°) a klesají směrem k nadiru ($-90°$)



Obrázek 3: Horizontální souřadnice

2.3.4 Převody mezi souřadnicovými soustavami

Když se bavíme o souřadnicovém systému, bavíme se vlastně o pohledu z určitého místa někam do prostoru. V prostoru si musíme určit nějaká jednotná pravidla, abychom jednoznačně dokázali určit pozici objektů. V kartézském systému máme počátek souřadnic a osy [x,y,z], které jsou na sebe kolmé. Pro některé aplikace se nám však hodí souřadnice kartézské převést na sférické. Hlavně, co se sférické astronomie týče, je jejich použití velmi výhodné a často na ně narazíte spíš, než na kartézské. Mezi souřadnicemi kartézskými a sférickými existuje oboustranný převodní vztah, pomocí kterého můžete pozice objektu převést z jednoho zápisu na druhý. [13]

Převod z kartézských souřadnic na sférické

$$r = \sqrt{x^2 + y^2 + z^2}$$
$$\theta = \arccos\left(\frac{z}{r}\right)$$
$$\varphi = \arctan\left(\frac{y}{x}\right)$$

kde:

θ ... sklon dráhy. Úhel svírající hlavní rovinu a objekt

φ ... azimut. Orientovaný úhel

Převod ze sférických souřadnic na kartézské

$$x = r\sin(\theta)\cos(\varphi)$$
$$y = r\sin(\theta)\sin(\varphi)$$
$$z = r\cos(\theta)$$

Při převodu mezi souřadnicemi ekliptickými a rovníkovými si vystačíme s rotační maticí, kde budeme rotovat kolem osy x [7]. Pomocí ní souřadnice rotujeme o úhel ϵ, což je sklon rotační osy a přibližně výše zmíněná hodnota $23°27'$ [4, str. 24]

Převod z ekliptikálních souřadnic na rovníkové

$$\begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix} = \begin{pmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & \cos(\epsilon) & -\sin(\epsilon) \\ 0.0 & \sin(\epsilon) & \cos(\epsilon) \end{pmatrix} \begin{bmatrix} x_e \\ y_e \\ z_e \end{bmatrix}$$

Pro převod z rovníkových souřadnic na ekliptické stačí matici transponovat.

Převod z rovníkových souřadnic na horizontální je odlišný v tom, že budeme potřebovat ještě argumenty navíc. Je to aktuální hvězdný čas a naše zeměpisná šířka. [9]

$$h = \arcsin\left(\sin(\delta)\sin(\varphi) + \cos(\delta)\cos(\varphi)\cos(t_{sidereal} - \alpha)\right)$$
$$A = \arccos\left(\frac{\sin(\delta) - \sin(\varphi)\sin(h)}{\cos(\varphi)\cos(h)}\right)$$

kde:

h ...výška

A ... azimut

δ ... deklinace

α ... rektascenze

φ ... zeměpisná výška pozorovatele

3 Výpočet polohy objektů

3.1 Planety

Souřadnice planet jsou v aplikaci vypočteny pomocí **teorie VSOP**. První verze teorie, VSOP82, byla vyvinuta **Pierrem Bratagnonem** v roce 1982 ve francouském institutu věd *Bureau des Longitudes*. Tato verze měla hlavní nedostatek v tom, že sada vstupních dat nemohla být zkrácená, pokud jsme chtěli rychlejší výpočet a stačily nám méně přesné výsledky. Tento nedostatek řeší verze z roku 1987 – VSOP87.

VSOP87 je jedním z nejpřesnějších výpočtů souřadnic a garantuje přesnost 1"$na\pm$4000 let od J2000.0. Nevýhoda VSOP (i v dnešní době) je ta, že postup výpočtu koordinátů je velmi špatně dokumentován a pro laika je obtížné najít ten správný. Mi se jej podařilo najít na serveru www.caglow.com [5], kde je algoritmus pěkně popsán.

Základem pro výpočet je dataset, který je zdarma dostupný ke stažení z FTP francouzského *Institut de mécanique céleste et de calcul des éphémérides*, který časem nahradil výše zmíněný *Bureau des Longitudes*. Nejedná se o jeden dataset, ale o celou sadu určených pro různé typy výpočtů.

- **VSOP87A** - heliocentrické ekliptikální kartézské souřadnice J2000.0
- **VSOP87B** - heliocentrické ekliptikální sférické souřadnice J2000.0
- **VSOP87C** - heliocentrické ekliptikální kartézské souřadnice data
- **VSOP87D** - heliocentrické ekliptikální sférické souřadnice data
- **VSOP87E** - barycentrické ekliptikální kartézské souřadnice J2000.0

Každá planeta má svůj soubor s daty. Tento soubor má název podle typu výpočtu a příponu podle zkratky planety. Například pro výpočet ekliptikálních kartézských souřadnice Venuše budeme hledat soubor VSOP87C.ven.

Použitý algoritmus v bakalářské práci pracuje s daty **VSOP87C**. Data jsou rozdělena do tří sekcí pojmenovaných X,Y,Z (VARIABLE 1-3). Uvnitř každé sekce jsou data rozdělena na sadu proměnných *T**O, *T**1. Každá sada obsahuje několik sloupečků čísel, kde poslední 3 nás budou zajímat. Pro budoucí výpočty si je pojmenujeme A,B,C. Tyto sloupce jsou použity k výpočtu jedné části (termu) celého vzorce. Tyto části jsou sečteny dohromady a jejich součty vloženy do rovnice pro zjištění aktuální polohy. Tento postup je shodný pro všechny koordináty a výsledek je přesnější s počtem vstupních dat.

$$T = (t_{\text{julian}} - 2451545)/365250$$
$$Term = A * \cos(B + C * T)$$

Proměnná Term se tímto způsobem vypočítá pro všechny řádky *T**0 a provede se jejich součet. Ve výsledku pro *T**0 dostaneme:

$$X_0 = \sum_{i=0}^{n} A_i \cos(B_i + C_i * T)$$

kde n je počet řádků v *T**0. T-hodnot je vždy 5. Pro souřadnici X tedy dostaneme proměnné X1,X2,X3,X4,X5. Výslednou souřadnici dostaneme takto:

$$X = X_0 + X_1 * T + X_2 * T^2 + X_3 * T^3 + X_4 * T^4$$

Tento postup zopakujeme pro souřadnice Y a Z. Výsledek dostaneme jako vzdálenost v **astronomické jednotce**. Vypočtené souřadnice jsou ekliptikální heliocentrické, proto se vztahují ke Slunci. Pokud chceme získat souřadnice geocentrické, musíme vypočíst heliocentrické souřadnice Země a o ně posunout koordináty dalších planet.

$$X_{\text{geocentric}} = X_{\text{planety}} - X_{\text{zeme}}$$
$$Y_{\text{geocentric}} = Y_{\text{planety}} - Y_{\text{zeme}}$$
$$Z_{\text{geocentric}} = Z_{\text{planety}} - Z_{\text{zeme}}$$

Nyní můžeme provést nějakou z transformací výše popsaných. Například převod do sférických ekliptikálních souřadnic nebo rovníkových.

3.2 Slunce

Při výpočtu pozice Slunce jsem vycházel z knihy **Astronomical Algorithms** [1, str.163]. Výpočet je přesný na 0.01°, což je pro naše potřeby dostačující. Pozice Slunce se dá vypočítat na základě pohybu Země, když se nebere v úvahu odchylka způsobena Měsícem a ostatními planetami. Tento výpočet je k nalezení ve zkrácené podobě také na webu www.geoastro.de [8]

$$
\begin{aligned}
M &= 357.52910 + 35999.05030 * jd_c - 0.0001559 * jd_c^2 - 0.00000048 * jd_c^3 \\
L &= 280.46645 + 36000.76983 * jd_c + 0.0003032 * jd_c^2 \\
C &= \left(1.914600 - 0.004817 * jd_c - 0.000014 * jd_c^2\right) * \sin(M) + \\
&\quad (0.019993 - 0.000101 * jd_c) * \sin(2M) + 0.000290 * \sin(3M))
\end{aligned}
$$

$$
\begin{aligned}
l &= M + C \\
\theta &= L + C \\
\alpha &= \arctan\left(\frac{\cos(\epsilon)\sin(l)}{\cos(l)}\right) \\
\delta &= \arcsin(\sin(\delta)\sin(l))
\end{aligned}
$$

kde:

M ... střední anomálie
L ... střední délka
C ... střed Slunce
l ... pravá délka
θ ... pravá anomálie
jd_c ... juliánské století

Výsledkem jsou rovníkové souřadnice II. druhu.

3.3 Měsíc

Výpočet souřadnic Měsíce je v aplikaci proveden podle algoritmu popsaného na webových stránkách www.stjarnhimlen.se [11]. Základní informace a český popis s mezivýpočty je také dostupný na kalendar.beda.cz [6]. Výpočet pozice Měsíce je oproti jiným objektům mírně složitější, jelikož se Měsíc nepohybuje po **rovině ekliptiky** a je také hodně ovlivňován Sluncem a dalšími tělesy. Níže popsaný postup výpočtu nám dává přesnost s možnou odchylkou větší, než 1°". Pro přesnější výpočty doporučuji ještě počítat s **perturbací**. Postup pro korekci souřadnic je uveden na výše zmíněných webových stránkách.

$$
\begin{aligned}
d &= \text{počet dní od 31.12.1999} \\
\Omega &= 125.1228 - 0.0.0529538083 * d \\
i &= 5.1454 \\
\omega &= 318.0634 + 0.1643573223 * d \\
a &= 60.2666 \\
e &= 0.054900 \\
M &= 115.3654 + 13.0649929509 * d
\end{aligned}
$$

$$
\begin{aligned}
E &= \text{eccentricanomaly} \\
x &= a * (\cos(ea) - e) \\
y &= a * \sqrt{1.0 - e^2} * \sin(E) \\
v &= \arctan(y, x) \\
r &= \sqrt{xv^2 + yv^2}
\end{aligned}
$$

$$
\begin{aligned}
\text{geocentric}_x &= r * (\cos(\Omega) * \cos(v + \omega) - \sin(\Omega) * \sin(v + \omega) * \cos(i)) \\
\text{geocentric}_y &= r * (\sin(\Omega) * \cos(v + \omega) + \cos(\Omega) * \sin(v + \omega) * \cos(i)) \\
\text{geocentric}_z &= r * (\sin(v + \omega) * \sin(i))
\end{aligned}
$$

Výsledkem jsou geocentrické ekliptikální souřadnice. Vzdálenost r je počítána v poloměrech Země.

3.4 Hvězdy

Hvězdy, oproti planetám, nevykonávají **keplerovský pohyb**, tj. pohyb po elipsách kolem jednoho hmotného tělesa. Znamená to také, že v konkrétní den jsou každý rok na *stejném místě*. Lépe řečeno, odchylka od roku předchozího je minimální. Odchylka může být způsobena například rozpínáním vesmíru - hvězdy a galaxie se od sebe vzdalují. Pro výpočet jejich souřadnic v čase se nám bude hodit aktualizovaný katalog hvězd, ve kterém najdeme jejich **rovníkové souřadnice**. My budeme vycházet z **Hipparcosova katalogu**, který je volně ke stažení. Zjednodušenou, zkrácenou a dobře strukturalizovanou podobu ve formátu JSON nabízí projekt SkyBeautiful [12]. Ten obsahuje více než 93000 hvězd, kde každá hvězda má svůj identifikátor (HIP), pozici, magnitudu, vzdálenost a barvu v B-V škále.

4 Jazyk Dart

V počátcích internetu a webu se stránky staticky servírovaly ze serveru uživateli. Později byla uživateli nabídnuta jakási dynamika v podobě skriptů na serveru, které sestavovaly webové stránky a vracely je prohlížeči. V roce 1995 prohlížeč **Netspace Navigator** přišel se zbrusu novým jazykem – **JavaScriptem**. Tento jazyk měl přivést do světa webu dynamiku na straně klienta. Znamenalo to rychlou odpověď z formuláře, na webu mohl existovat skrytý obsah, který se zobrazil po kliknutí na záložku, a vůbec dostal internet do rukou nástroj, jak omezit komunikaci se serverem a ušetřit tak čas s přenášením dat.

JavaScript ve svých počátcích sloužil především jako nástroj pro drobné úpravy a vylepšení stránky vygenerované skriptem na serveru. Bylo snadné jej začít používat, jelikož jeho syntaxe nebyla nikterak obtížná, skripty byly znovupoužitelné a webem se začaly šířit návody jak danou věc v JavaScriptu naprogramovat. Nikdo si však tenkrát nedokázal představit, že by se v něm psaly celé webové aplikace a komplikovanější nástroje typu textový nebo grafický editor. Internet, jako každý *živý organismus* prochází svou evolucí. S příchodem rychlejšího připojení, vyššího výpočetního výkonu a lepší grafiky se člověk začal ptát „Proč bychom nemohli použít JavaScript pro vývoj komplexnějších aplikací?“ První odvážlivci se vrhali do neznámých vod s tím, vytvářet více a více interaktivnější a rychlejší aplikace. Z počátku ležela pořád téměř veškerá aplikační logika na serveru, ale vyvinul se AJAX (asynchronní JavaScript a XML), který dovoloval uživateli bez obnovení stránky poslat dotaz na server a na pozadí získat zpět odpověď. Díky nástupu nových technologií došlo ke snaze přesunout co největší část aplikační logiky ke klientovi. Prohlížeče podporující HTML5 umí pracovat s binárními daty, obrazem, zvukem nebo dokonce geolokací. Vznikají nové verze AJAXu a nové možnosti, jak komunikovat se serverem – například pomocí technologie **Web Sockets**, která nabízí oboustrannou komunikaci mezi serverem a klientem. Za těchto podmínek nebrání vývojářům webových aplikací nic v tom spouštět bohaté aplikace přímo u uživatele a nabídnout mu tak komfort rychlé odezvy a dynamického uživatelského rozhraní.

Jazyk JavaScript je poměrně jednoduché se naučit, je však obtížné se jej *naučit dobře*. Je to z toho důvodu, že s sebou občas přináší zvláštní chování, nejasnou syntaxi a pozůstatky z dob minulých. To celé ještě umocňuje různorodá podpora prohlížečů. Není to tak, že by JavaScript v prohlížečích chyběl, je to spíš různá interpretace jeho kódu – jeho pomyslná Achillova pata. V dnešní době však existují knihovny typu jQuery nebo **RIA frameworky** jako například Dojo. Tyto pomáhají vývojáři oprostit se od potřeby testovat a upravovat svůj kód pro různé prohlížeče a krkolomné konstrukce převádí na snadno zapamatovatelné funkce. Oproti jiným jazykům má JavaScript jisté nedostatky, které mu mohou bránit a brání v rozvoji.

- Příkaz není nutné zakončit středníkem
- Používají se prototypy, ne třídy. Objekty je možné vytvořit několika různými způsoby, které nejsou vždy úplně správné. Ve výsledku to může v kódu způsobit nestandardní chování.
- Pro začátečníky je velmi snadné ztratit aktuální kontext (this).

- Dědičnost je kapitola sama o sobě.
- Problémy s udržením závislostí skriptů a importu podpůrných knihoven.
- Nízký výkon.
- Již zmíněný problém s nekompatibilitou v prohlížečích.
- A další...

V důsledku těchto nedostatků bylo jen otázkou času, než hlavní lídři v oblasti internetových aplikací přijdou s vlastním řešením. Společnost **Google** proto v roce 2011 přišla s novým webovým jazykem – Dartem.

Dart není pouze jazyk, ale celé prostředí nástrojů, které dostane programátor do rukou. Jsou to především standardní knihovny, usnadňující práci s HTML, asynchronními událostmi, matematiku atd. Je to například Dart Editor, virtuální stroj nebo kompilátor Dartu do JavaScriptu – **dart2js**. Jedná se o jazyk s přehlednou a jasnou syntaxí, kterou známe z C++ nebo Javy.

4.1 Specifika jazyka

Dart je moderní jazyk, který s sebou, jako mnoho jiných, nese specifické vlastnosti a konstrukce. V úvodu je vhodné říci, že všechno je objekt. Dokonce i čísla, funkce a null. Všechny objekty dědí ze třídy **Object**. Podobně jako JavaScript, Dart umožňuje vytvářet zanořené (nested) funkce a platí pro něj podobná pravidla pro vytváření **scope**. Zde je však velký rozdíl v přehlednosti kódu a není možné tak snadno ztratit kontext, jako v Javascriptu. Podle konvence se názvy tříd píší v *UpperCamelCase*, názvy proměnných *lowerCamelCase* a názvy knihoven *lower_snake_case*.

Knihovny a importy pomáhají, oproti JavaScriptu, udržet si závislosti a importovat pouze potřebné moduly a komponenty. Po stažení instalačního balíčku získáte do rukou SDK se základními knihovnami, jako je práce s html, matematikou nebo asynchronním voláním. Další knihovny lze snadno získat pomocí balíčkovacího systému PUB (kapitola 4.2). Mějme soubor *mujprojekt.dart*, který vypadá takto:

Zdrojový kód 2: Ukázka importu knihoven

```
// Vytvorime knihovnu mujprojekt
library mujprojekt;

// Import knihovny async z SDK Dartu
import 'dart:async';
// Skript z knihovny Knihovna
import 'package:Knihovna/skript.dart';

// covertor.dart je cast knihovny
part 'package:MujProjekt/image_convert.dart';

void main() {...}
```

Řekněme, že skript *mujprojekt.dart* bude spouštět aplikaci. Pro naše účely jsme si napsali převodník obrázků z jednoho formátu do jiného. Tento převodník bude v adresáři *lib/image_ convert.dart*. Jelikož se jedná o část knihovny mujprojekt, na začátku souboru *image_ convert.dart* to musíme říct.

Zdrojový kód 3: Hlavička části knihovny

```
part of mujprojekt

class ImageConvert {...}
```

Nepovinné datové typy jsou jedna z klíčových vlastností Dartu. Datové typy v JavaScriptu jako takové neexistují, proto může být u větších projektů problém udržet čitelnost kódu. Další nevýhodou beztypových jazyků bývá často malý výkon z důvodu náročnější práce s pamětí a zjištění typu. Typovaný kód se dá lépe optimalizovat, což je jeden z důvodu, proč má JavaScript získaný kompilátorem dart2js lepší výkon. Zajímavé je také, že existuje datový typ **num**, který zahrnuje jak desetinná, tak celá čísla. Hodí se jej použít například tam, kde neznáme předem datový typ čísla. Naopak, pokud použijeme **int** nebo **double**, získáme lepší výkon.

Jednořádkové funkce nám pomáhají s přehledností kódu. Pokud máme například metodu, která vrací nějaký jednoduchý matematický výpočet, můžeme zápis zkrátit ze složených závorek a příkazu return na => (zdrojový kód 4). S tímto zápisem se často setkáme u lambda funkcí.

Properties jsou funkce, které se z pohledu přístupu k objektu tváří jako jeho atributy. Použít properties je vhodné, když má například objekt typu **Ctverec** vrátit svůj obsah vypočítaný z délky strany. V tom případě budeme mít property **Ctverec.obsah**. V Dartu ji vytvoříme přidáním klíčového slova **get** a **set** k názvu funkce. Property najdeme často ve spojení s jednořádkovými funkcemi.

Zdrojový kód 4: Ukázka použití properties

```
class Ctverec {
   double strana;
   double get obsah => strana*strana;
}
```

Futures jsou datovým typem, který *dává slib* že v budoucnosti funkce něco vrátí. Jedná se o náhradu callbacků, které v JavaScriptu často způsobovaly nesrozumitelné zanoření kódu a přispívaly k menší čitelnosti a špatné logice.

Streams jsou proudy dat, které programátor naslouchá a snaží se zachytit. Stream je například HTML událost – kliknutí na element, rolování kolečkem, či stisk klávesy. Čtení ze souboru vrací taktéž Stream.

Named contructors neboli pojmenované konstruktory. Jedná se vlastně o přívlastek konstruktoru, který mu dá programátor, aby v kódu ještě více specifikoval, jak chová. Mějme například třídu **DisplayMessanger**, která nám bude udržovat informaci o tom, co se vykreslí na obrazovku. Při vytvoření objektu budeme chtít, aby se všechny atributy nastavily na *true*. Aby bylo jasné, co konstruktor provede, přidáme mu přívlastek .enableAll().

Zdrojový kód 5: Pojmenované konstruktory

```
class DisplayMessanger {
    bool showStars, showAzimuths, showAltitudes, showPlanets;

    DisplayMessanger.enableAll() {
        showStars = true;
        showAzimuths = true;
        showAltitudes = true;
        showPlanets = true;
    }
}

DisplayMessanger dm = new DisplayMessanger.enableAll();
```

Cascade operator souží k řetězení volání metod a atributů jednoho objektu. Jedná se o zajímavou vlastnost podporující lepší čitelnost a srozumitelnost kódu. Zápis se provádí pomocí dvou teček (..) místo jedné, jako při běžném volání.

Zdrojový kód 6: Cascade operator

```
objekt.lehni().skoc();

objekt
    ..lehni()    // metoda se zavola na objektu "objekt"
    ..skoc()     // metoda se zavola na objektu "objekt"
```

this v konstruktoru nastavuje inicializační proměnné. Nepotřebujeme v parametrech konstruktoru přijímat data do proměnných, a ty přiřadit až atributům objektů. Dart toto přiřazení usnadňuje a za pomocí klíčového slova **this** v parametru můžeme atribut rovnou nastavit.

Zdrojový kód 7: Použití this v konstruktoru

```
StencilBuffer {
    int with, height;
    StencilBuffer(this.width, this.height) {
        canvas = new CanvasElement(width: width, height: height);
        context = canvas.context2D;
        context.imageSmoothingEnabled = false;
    }
}
```

Volitelné parametry můžeme rozdělit do dvou kategorií. Jsou to pojmenované a nepojmenované volitelné parametry. První z nich jsou identifikovány podle pořadí ve funkci, druhé podle jména při volání. Tato vlastnost je podobná, jako v jazyku Python s tím rozdílem, že není povoleno použít poziční a pojmenované argumenty v jedné sadě parametrů.

Zdrojový kód 8: Použití volitelných parametrů

```
void vlozCloveka(String jmeno, String prijmeni, [String pohlavi]) {...}
vlozCloveka("Ladislav","Smoljak");
vlozCloveka("Ladislav","Smoljak","muz");

void vlozCloveka(String jmeno, String prijmeni, {String pohlavi}] {...}
vlozCloveka("Tomas","Bata");
vlozCloveka("Tomas","Bata",pohlavi: "muz");
```

4.2 Správce softwarových balíčků pub

pub je systém pro správu balíčků, knihoven třetích stran. Při psaní aplikace v JavaScriptu programátor často narazí na problém se závislostmi. Chce využít jednu knihovnu, která řeší konkrétní problém, ale ta využívá některých vlastností z knihovny jiné. Vývojář musí tyto závislosti nalézt a udržovat. Dalším problémem mohou být různé verze knihoven, které nejsou vzájemně kompatibilní. Projekt v Dartu obsahuje soubor **pubspec.yaml**, do kterého přidáme knihovny, které chceme využít. Jedná se o klasický textový soubor s **YAML syntaxí**.

Zdrojový kód 9: Ukázka souboru pubsec.yaml

```
name: SkyMap
author: Jiri Kupka
dependencies:
  browser: any
  intl: any
  vector_math: 1.4.2+1
```

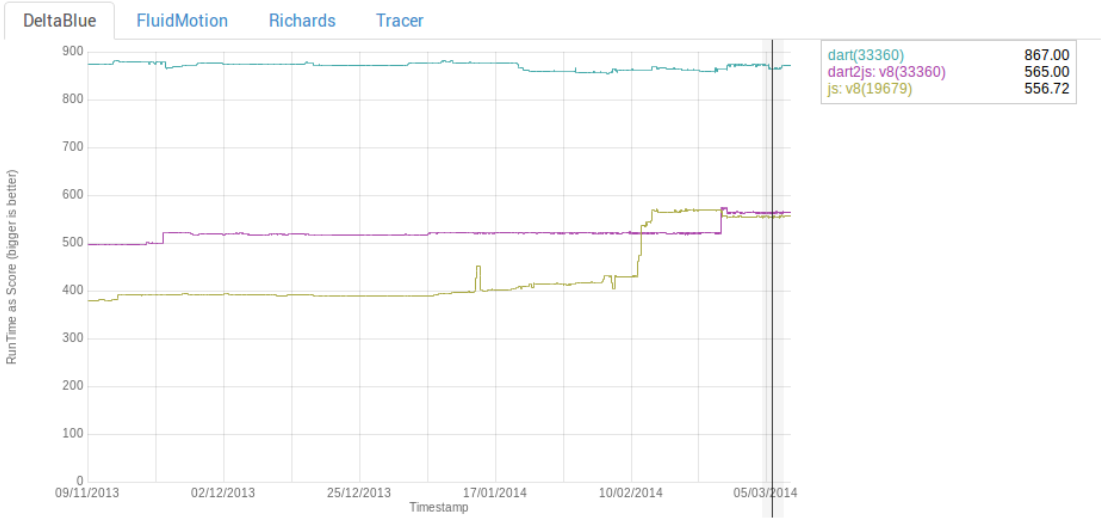
V sekci dependencides vidíme například závislost na balíčku **vector_math**, který obsahuje třídy typu Vector nebo Matrix. Balíček vector_math obsahuje taky svůj vlastní soubor **pubspec.yaml**, ve kterém jsou zapsány závislosti potřebné pro jeho běh. Toto nám zaručuje, že když spustíme nástroj **pub get**, rekurzivně se projdou všechny pubsec soubory a stáhnou se potřebné balíčky. U každého balíčku můžeme taky určit konkrétní verzi, využít operátorů >,<,= nebo pomocí **any** říci, že nám je jedno, která verze se použije. Chytrým řešením je také, že se knihovny stáhnou pouze jednou do speciálního adresáře na disku a všechny projekty, které knihovnu využívají, se na ni odkazují pomocí symbolického odkazu na souborovém systému.

4.3 Transpiler dart2js

Dart je nový jazyk a jeho největší slabost je ta, že nativní podpora v prohlížečích není žádná. V jeho instalačním balíčku nalezneme prohlížeč **Dartium**, což je vlastně Chromium

s virtuálním strojem, který umí spustit kód v Dartu. Zakomponování virtuálního stroje do prohlížeče Chrome se chystá v blízké době, otázkou je však, jak na tom budou ostatní prohlížeče. Apple s Microsoftem jsou toho názoru, že do svých prohlížečů nikdy Dart VM neimplementují. Jsme však na úplném začátku jeho existence a možná vše dopadne jinak. Silným argumentem, proč Dart do prohlížečů implementovat, je výkon. JavaScript naráží na určité hranice, jeho výkon se sice mírně zlepšuje, ale v porovnání s Dartem je nízký.

Aby bylo možné v Dartu psát aplikace i bez potřeby mít v prohlížeči virtuální stroj, vznikl kompilátor do JavaScriptu – dart2js. Zajímavou vlastností je, že si kompilátor může oproti programátorovi dovolit ve výsledném kódu provádět takové konstrukce, které by si vývojář nedovolil. Výsledek je takový, že zkompilovaný kód je mnohdy ještě rychlejší, než ten napsaný programátorem v čistém JavaScriptu. Úplný graf výkonu nalezneme na webových stránkách Dartu. Díky překladači máme možnost mít naši aplikaci napsanou jak v Dartu, tak v JavaScriptu. Pokud prohlížeč nebude mít zabudovaný virtuální stroj, spustí se klasicky kód v JavaScriptu.



Obrázek 4: Srovnání výkonu v benchmarku DeltaBlue [15]

5 Implementace hvězdné mapy

Před samotným začátkem implementace hvězdné mapy bylo důležité nastudovat základní výpočty a pojmy sférické astronomie. Chtěl jsem, aby aplikace vykreslovala nejjasnější hvězdy, planety Sluneční soustavy, Slunce a Měsíc. Bylo poměrně snadné odhadnout, že výpočet souřadnic hvězd bude jednodušší, než souřadnic planet a jiných blízkých objektů. Problémem však bylo najít ten správný postup. Díky dostupnosti internetu má každý z nás možnost dostat se k informacím, které by v minulosti musel složitě hledat a shánět v hůře

dohledatelných zdrojích. Je tak mnohem snadnější získat potřebné informace a využít. Po získání správných algoritmů bylo nutno rozhodnout, jakým způsobem se budou objekty vykreslovat. Byla zde možnost využít **Canvas2D** nebo **WebGL**. WebGL by možná nabízelo lepší výkon s podporou hardwarové akcelerace v prohlížeči, ale přineslo by také větší složitost samotného zdrojového kódu. Navíc Canvas2D je o něco starší technologie, tudíž nabízí vyšší podporu ve starších prohlížečích.

Dalším otazníkem bylo využití některé z již dostupných knihoven Dartu pro práci s grafikou a animacemi. Zajímavou knihovnou pro grafiku a algebru je **Vector Math**. Najdeme v ní třídy pro práci s vektory a maticemi, což je vhodné pro reprezentaci souřadnic a jejich transformace. Přemýšlel jsem také nad využitím knihovny **StageXL**, která obsahuje funkce a třídy pro 2D grafiku, animaci, zvuk a hry. Samotná knihovna je jakýmsi *portem* ActionScriptu z Flashe do prostředí Dartu a HTML5. Nakonec jsem knihovnu nevyužil, jelikož jako mnoho dalších programátora odstiňuje od základních principů dané problematiky a obaluje základní výchozí konstrukce svými. Chtěl jsem si vyzkoušet, jak přesně fungují herní smyčky, identifikace objektů, nebo jak optimalizovat kód a získat tak lepší výkon.

Poslední částí před samotným začátkem programování bylo zvolení vhodné architektury kódu. Jak správně rozdělit kód, aby byl znovupoužitelný a další rozvoj byl co nejjednodušší. Již dávno jsou pryč doby, kdy se psal kód sekvenčně (imperativně), nejlépe do jednoho souboru a bez tříd. Díky OOP máme k dispozici silné vlastnosti, jako je zapouzdření nebo polymorfizmus. I přesto se dá psát *špagetový kód*, který vzniká absencí pravidel a architektury. Dnes je zřejmě nejvíce známá architektury **MVC**, která kód logicky rozděluje do tří vrstev. Tyto vrstvy jsou zaměnitelné a pomáhají s udržením kvalitního kódu.

5.1 Model MVC

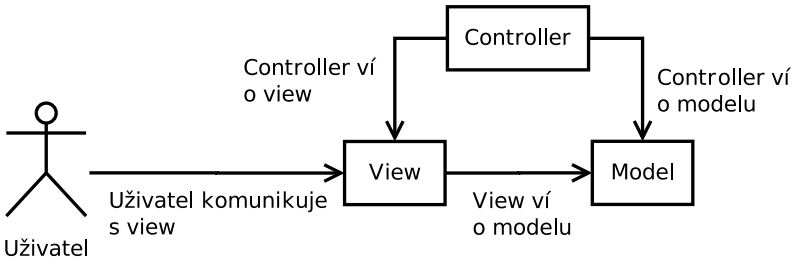
Pro implementaci samotné aplikace jsem vycházel z návrhového vzoru/softwarové architektury MVC. Tak jako mnoho aplikací a programátorů, i já jsem si přizpůsobil model MVC k obrazu svému, principy však zůstaly stejné. Architektura je rozdělena na 3 části - Model, View a Controller.

Model má na starosti veškerou aplikační logiku a je možné jej vyjmout a použít nezávisle od View nebo Controlleru. Najdeme zde funkce pro načtení, zpracování a uložení dat získaných z katalogů nebo pomocí výpočtu.

View má na starosti zobrazení dat uložených v modelu uživateli. V konstruktoru je mu předán odkaz na vytvořený model. View vytváří uživatelské prostředí, jehož základem je canvas. Existuje zde hlavní funkce **redraw()**, po jejíž zavolání se získají aktuální data z modelu a ty se vykreslí do vytvořeného Canvasu. View jako takové se dá vyjmout a nahradit za jiné. Například s technologií WebGL by se dalo vytvořit *náhradní* view, které by objekty vykreslovalo do 3D prostředí.

Controller je nejošemetnějším místem aplikace. U controlleru, jak už to bývá, je těžké určit jeho hranice. Startuje zde samotná aplikace – vytváří se instance modelu a view. Já

jsem si controller zvolil za místo, kde do aplikace vstupují vnější události. Najdeme zde proto dva časovače, které nezávisle na sobě volají funkce pro výpočet nových souřadnic objektů a jejich vykreslení. Okolním vlivem je i geolokace, která, pokud je to možné, je získaná z API prohlížeče. Jeho volání rovněž najdeme v controlleru. Události vyvolané uživatelem jsou v aplikaci zachyceny a zpracovány přímo ve view, stejně tak jako prezentační logika.



Obrázek 5: Architektura aplikace, MVC

5.2 HTML5 Canvas

Canvas, neboli *plátno* je jeden z elementů, které se nově vyskytují v HTML5. Do canvasu se pomocí javascriptu vykresluje bitmapová grafika, což vede k vývoji her a aplikací ve webovém prohlížeči. Pokud chtěl programátor dříve psát webové hry a aplikace, měl poměrně omezené možnosti. Musel si vystačit pouze s HTML jako takovým a každý grafický prvek byl vlastně HTML element. Canvas je blokový element, který na stránce vytvoří *plochu* pro vykreslení grafiky. Tento element má metodu **getContext()**, pomocí které získáme *kontext*, s kterým už můžeme operovat. Jako parametr metody getContext() se vkládá **contextId**, což je vlastně typ kontextu. Prohlížeče mohou implementovat různé typy kontextů s různým API. Dnes jsou nejznámější kontexty **2d** a **webgl**, případně **experimental-webgl**. V aplikaci používám kontext **2d**, který zpřístupňuje API pro práci s 2D grafikou.

Do získaného kontextu můžeme vykreslovat obrázky různými způsoby. Prvním způsobem je volání jeho kreslících metod. Jsou to metody vykreslující jednoduché tvary (kruh, čtyřúhelník) nebo křivky.

Zdrojový kód 10: Ukázka volání metod HTML5 canvasu

```
var context = canvas.getContext('2d');
context.beginPath();          // Zacatek cesty
context.moveTo();             // Presuneme "stetec" na pocatek cesty
context.lineTo(x,y);          // Tahneme "stetec" na pozici [x,y]
context.closePath();          // Uzavreme cestu
context.stroke();             // Vykreslime
```

Tvar vykreslený pomocí kontextu má svou barvu výplně a okraje. Barvu můžeme zadávat v paletě RGB, RGBA jak v hexa podobě, tak decimální. Navíc, hexa zápis může být

i ve zkráceném tvaru. Jedná se tedy o velmi tolerantní zápis podobný CSS. Zadání tloušťky čáry nebo velikosti písma je hodně podobný a opět pravděpodobně inspirovaný CSS.

Zdrojový kód 11: Styl vykreslování do HTML5 canvasu

```
context.fillStyle = "#f0f";    // Barva vyplne bude odted fialova
context.fillStroke = "#f00";   // Barva ohraniceni bude cervena
context.font = "16px Arial";   // Pismo bude Arial o velikosti 16px
```

Kromě metod pro generování grafiky můžeme do canvasu vložit i vlastní obrázek, například z HTML elementu <img> nebo <video>. Takto vložený obrázek/snímek videa teď v canvasu znamená pole RGBA pixelů, které tvoří samotný obrázek. Programátor může tyto pixely získat pomocí metody **getImageData()** a provést jejich modifikaci, aplikovat filtr, obecně zpracovat obrazovou informaci. Modifikované data, stejně jako jsme je získali, můžeme do canvasu zpět zapsat. A to pomocí metody **putImageData()**. Tímto se nám dostává do rukou nástroj pro tvorbu multimediálních aplikací, nebo dokonce webových aplikací z oblasti zpracování obrazu.

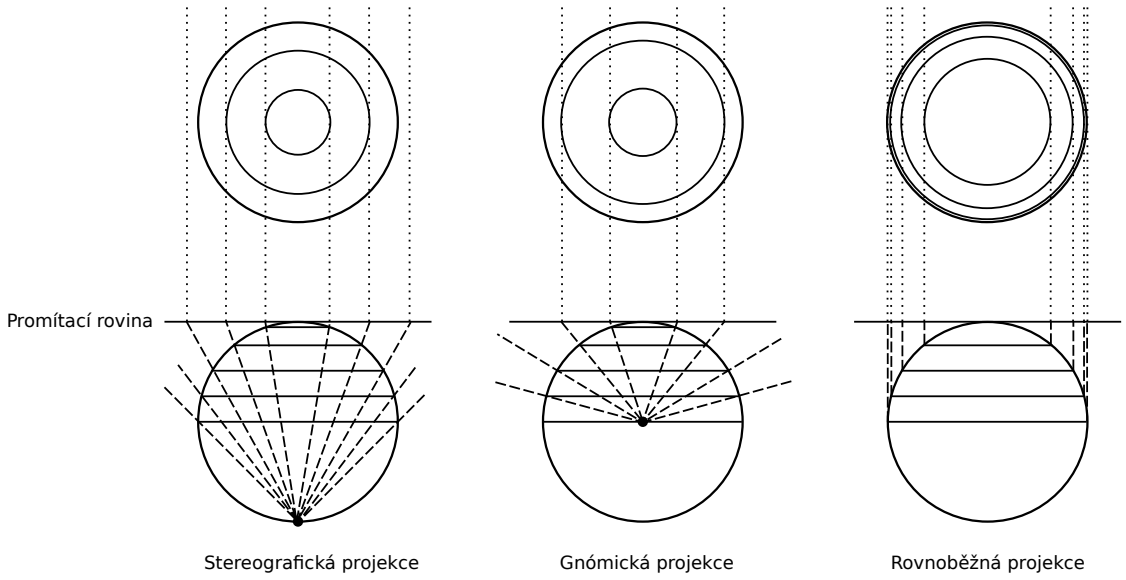
Metody pro práci s bitmapovou grafikou nejsou jediné, které Canvas API nabízí. Najdeme zde ještě funkce a proměnné nastavující jeho chování, jako například atribut **imageSmoothingEnabled**, kterým můžeme zakázat/povolit anti-aliasing, **globalAlpha** upravující průhlednost a další.

Context 2D je prostředí, ve kterém jsou obrazová data uložena. Toto prostředí je tvořeno objekty uloženými v síti [x,y] a transformační maticí. Matice je ve výchozím stavu jednotková, takže *„kde vložíte objekt, tam jej najdete“*. Matici si však můžete upravit pomocí metod scale(), translate(), transform() a dalších. Důležité jsou také funkce save() a restore(). Pomocí první z jmenovaných uložíte stávající transformační matici do paměti. V tomto okamžiku můžete provést transformace s vykreslenými objekty a následně pomocí metody restore() matici navrátit do původního stavu.

5.3 Stereografické promítání

Získané souřadnice objektů musíme uživateli nějakým způsobem zobrazit. Nabízí se několik možných řešení. V nejjednodušším případě bychom mohli uživateli horizontální souřadnice zobrazit do dvou kruhů reprezentujících horní polokouli (směrem k **zenitu**) a dolní polokouli (směrem k **nadiru**). Toto řešení však moc nevyhovuje interaktivní aplikaci a lidskému vnímání prostoru. Máme proto sadu mapových projekcí, pomocí kterých můžeme souřadnice lépe zakreslovat. Různé typy projekcí se hodí pro různé účely. Jednoduché projekce můžeme rozdělit na **azimutální**, kde se souřadnice zobrazují přímo na rovinu, **kuželová**, které zobrazují na povrch kužele a **válcová** zobrazující souřadnice na povrch válce. Takový objekt se poté *rozvine* do roviny a získá se mapa. Nejčastěji se používá azimutální projekce, která se dále dělí podle toho, odkud se promítá. Máme **gnómickou projekci**, která promítá ze středu objektu (v našem případě Země), **stereografickou projekci**, promítající z protějšího pólu a **ortografickou** (obr. 6), která promítá vodorovnými paprsky z nekonečna.

Nejvhodnější se mi zdálo použití **stereografické projekce** [14], která je známá již dlouhou dobu a pravděpodobně poprvé vzešla z hlavy již zmíněného Hipparchose. Stereografická projekce je použita například v astrolabu nebo v ciferníku **pražského orloje**. Zajímavou vlastností tohoto zobrazení je, že zachovává úhly. Daří se tak pomocí ní jednoduše provádět astronomické výpočty.



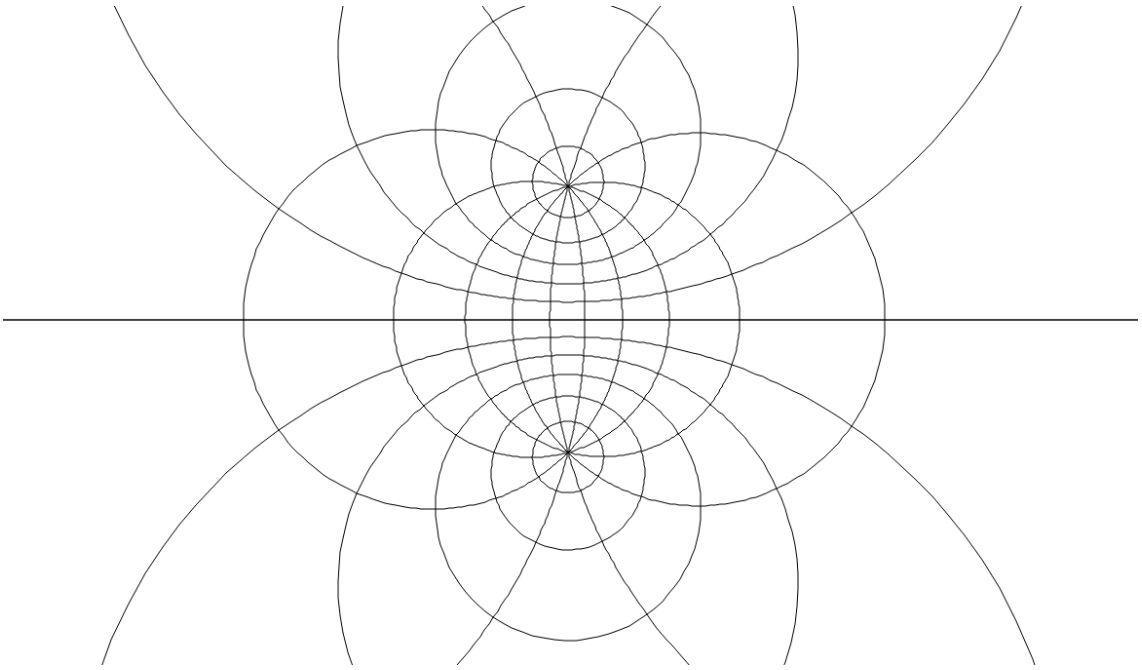
Obrázek 6: Typy azimutálních projekcí

Zobrazovací rovnice

$$k = \frac{2R}{1 + \sin(\phi_1)\sin(\phi) - \sin(\phi_1)\cos(\phi)\cos(\lambda - \lambda_0)}$$
$$x = k\cos(\phi)\sin(\lambda - \lambda_0)$$
$$y = k\left[\cos(\phi_1)\sin(\phi) - \sin(\phi_1)\cos(\phi)\cos(\lambda - \lambda_0)\right]$$

kde:

R ... poloměr sféry
ϕ ... šířka tělesa
ϕ_1 ... šířka středu
λ ... výška tělesa
λ_0 ... výška středu



Obrázek 7: Stereografická projekce

5.4 Identifikace objektů

Jedna z vlastností aplikace je ta, že po kliknutí na vykreslený objekt se uživateli o něm zobrazí informace, jako jsou jeho koordináty nebo, v případě hvězd, jejich magnituda. Nevýhoda canvasu je ta, že objekty do něj vykreslené nemají žádný identifikátor. V případě HTML můžeme prakticky každý vykreslený element identifikovat v jeho stromové struktuře, v případě canvasu to však neplatí. Canvas je pouze plátno, do kterého se vykreslují jen obrazová data – informace o barvě určitého pixelu. Pokud chceme identifikovat objekt, který leží na určité pozici, musíme si ukládat informaci o tom, že právě na této pozici jsme ho vykreslili. V aplikaci jsou všechny objekty, které se vykreslují, uložené v seznamu. V tomto případě můžeme jako jedinečný identifikátor objektu využít jeho pozici v poli. Přístup k takovémuto objektu bude mít složitost $O(\mathrm{N})$. V tomto okamžiku umíme na základě indexu získat objekt, který reprezentuje vykreslené těleso. Nyní ještě musíme zaručit, abychom po kliknutí na obrazovku tento index získali. Nejjednodušší řešení, které jsem nejdříve aplikoval je to, že jsem si vytvořil dvourozměrné pole o velikosti obrazovky, do kterého jsem při vykreslování objektu na pozici [x,y] vložil do indexů [y][x] jeho index. Takový kód vypadal zhruba takto:

Zdrojový kód 12: Ukázka triviálního stencil bufferu

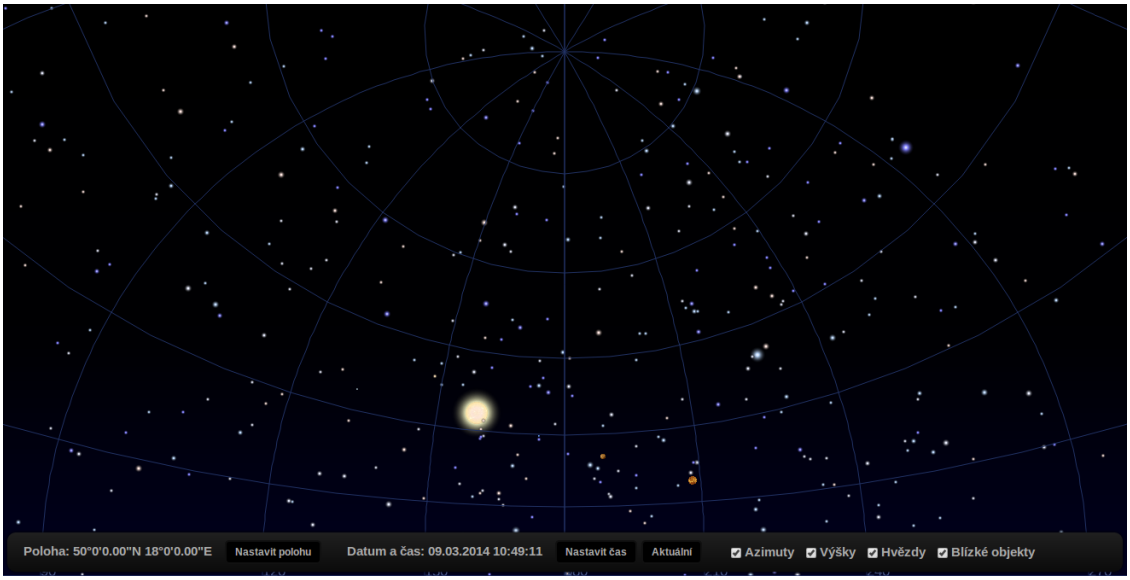
```
int index = 0;
buffer.clear();
objects.forEach((CelestialObject o) {
    // Vykresleni objektu do canvasu
    buffer[y][x] = index++;
});
```

V tomto případě by se mohlo zdát, že je všechno v pořádku, bohužel je zde několik nedostatků. Tento kód jednak nebere v úvahu velikost objektu a jeho tvar, ale pouze bod reprezentující jeho střed. Další nevýhodou je mazání, které nemusí být efektivní. Každý vykreslený objekt má tvar kruhu s poloměrem několik pixelů. Kruh tvoří množina pixelů a my potřebujeme, aby uživatel, pokud klikne kdekoli v rámci jeho plochy, bez většího *mířen*í, o něm získal informace. Takto musíme obalit buffer do objektu, který bude index zapisovat nejen do středu objektu, ale přes celý jeho poloměr. Tady narážíme na neefektivitu zápisu do pole. Čtení je rychlé, ale zápis ve spojení s mazáním velmi pomalý. V reálném čase, kdy potřebujeme mít plynulé uživatelské rozhraní, toto není možné udělat. Při řešení tohoto problému jsem se inspiroval v OpenGL. V OpenGL existuje **stencil buffer**, což je paměť, do které se na pozici vykresleného pixelu uloží číselná hodnota. V projektu tuto paměť simuluji sekundárním canvasem, do kterého objekt ještě jednou vykreslím. Podstata tkví v tom, že barva tohoto objektu je vlastně jeho zakódovaný index. V projektu předpokládám, že budu pracovat s méně, než 65 536 (256^2) objekty a barvu budu ukládat do RGB. Místo, kde není žádný objekt je bílé (má RGB(255,255,255)). Místo, kde leží nějaký objekt má svou barvu uloženou následovně:

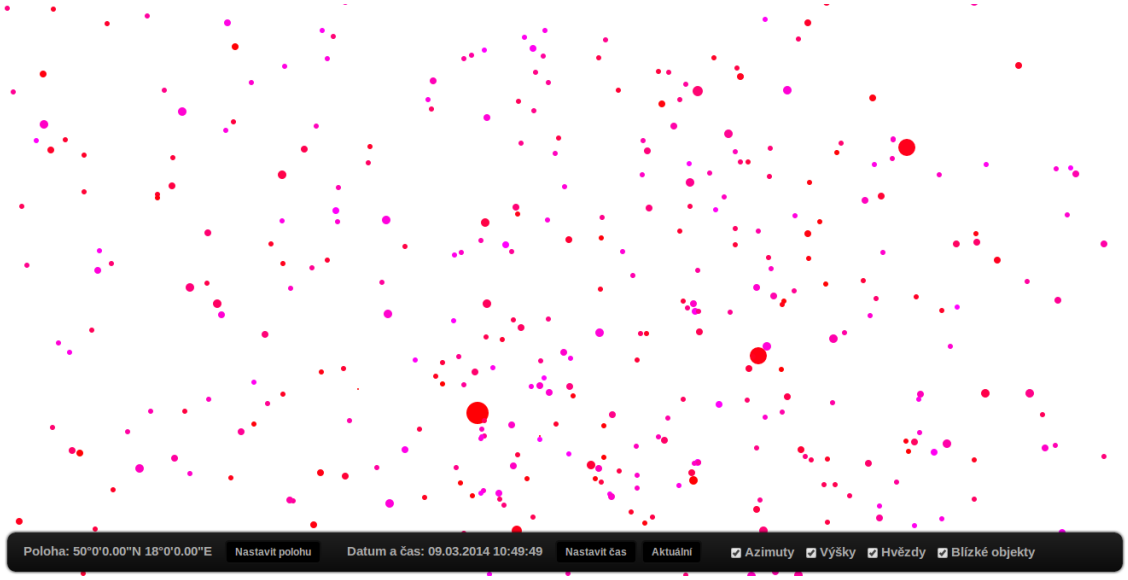
```
RGB = (255,INDEX~/255,INDEX\%255)
```

kde operátor / je celočíselné dělení.

Jelikož v tomto případě pracuji přímo s optimalizovanými funkcemi canvasu, jako jsou metody arc() pro vykreslení kruhu do plátna nebo fillRect() pro vymazání celého plátna při obnově, získám vysoký výkon. O něco pomalejší je získání těchto dat zpět, protože se obrázek musí převést na pole bytů a dopočítat index v poli z kliknutého místa. Vzhledem k tomu, že uživatel chce získat informace řádově méně častěji, než potřebuje tyto informace zapsat, je tento způsob uložení velmi výhodný.



Obrázek 8: Mapa hvězdné oblohy z pohledu uživatele



Obrázek 9: ID objektů zakódované do RGB

5.5 Přehled knihoven aplikace

- **skymap_app.dart** controller. Obsahuje hlavní třídu **SkyMapApp**.
- **skymap_model.dart** model aplikace. Obsahuje doménovou logiku a práci s datovými zdroji.
- **skymap_view.dart** obsahuje view třídu vykreslující data z modelu.
- **widgets.dart** obsahuje sadu komponent/přídavných modulů pro view. View vytváří instance komponent a v každé iteraci cyklu zobrazení volá na widgetu metodu redraw(), čímž informuje widget, že pokud vykresluje data z modelu, může je překreslit.
- **graphics.dart** obsahuje třídy pro práci s grafikou. Například implementaci stencil bufferu.
- **objects.dart** je jednou z nejdůležitějších částí. Obsahuje třídy reprezentující nebeské objekty.
- **geo.dart** obsahuje třídu přenášející informaci o pozici pozorovatele.
- **astro_functions.dart** se skládá ze sady funkcí pro jednoduché astronomické a geometrické výpočty. Převod mezi souřadnicovými systémy nebo získání sklonu rotační osy. Najdeme zde taky astronomické konstanty, jako je délka světelného roku nebo astronomické jednotky.
- **datetime.dart** obsahuje třídy pro práci s časem. Hlavní třídou zde je **JulianDate**.
- **settings.dart** obsahuje nastavení konstant pro běh aplikace. Najdeme zde cesty k obrázkům planet nebo nastavení výchozí zeměpisné šířky a výšky pozorovatele.
- **dictionary.dart** se skládá ze slovníku, odkud aplikace čerpá popisné texty pro uživatele.

5.6 Přehled tříd aplikace

- **SkyMapApp** existuje právě jedna instance této třídy. Zavoláním metody **run()** se spustí výpočet souřadnic a zobrazování objektů. Třída obsahuje odkazy na model a view.
- **SkyMapModel** model obsahující doménovou logiku a přístup ke zdrojům aplikace. V systému by měla existovat právě jedna instance, kterou udržuje třída SkyMapApp. Instance modelu je předána pomocí controlleru třídě SkyMapView. Obsahuje metodu **compute()**, která provede výpočet aktuálních souřadnic objektů.
- **SkyMapView** view vykreslující objekty do canvasu. Udržuje odkaz na model a čerpá z něj data pro vykreslení objektů. V systému by měla existovat právě jedna instance této třídy vytvořená controllerem. Třída obsahuje metodu **redraw()** překreslující canvas.

- **DisplayMessanger** pomocná *přepravka* udržující část stavových informací prezentační logiky.
- **ControlWidget** abstraktní třída, ze které dědí všechny widgety. Obsahuje metodu **redraw()** volanou ze třídy SkyMapView.
- **PositionControlWidget** widget pro nastavení pozice pozorovatele uživatelem.
- **TimeControlWidget** widget pro nastavení pozorovaného času uživatelem.
- **DisplayControlWidget** widget umožňující uživateli zvolit, které prvky hvězdné oblohy se mu budou zobrazovat.
- **ObjectInfoControlWidget** widget zobrazující okno s podrobnými informacemi o poloze objektu po kliknutí na něj.
- **FinderWidget** umožňuje vyhledat objekt pomocí jeho názvu.
- **CelestialObject** abstraktní třída, ze které dědí všechny další třídy prezentující vesmírné objekty. Obsahuje metody a atributy, které musí všichni potomci implementovat. Je to například získání horizontálních souřadnic nebo identifikátoru objektu. Důležitá je zde metoda **compute()**, která se volá z modelu a počítá nové souřadnice daného objektu.
- **Planet** třída reprezentující planetu Sluneční soustavy. V metodě **compute()** se aktuální souřadnice počítají pomocí teorie VSOP87.
- **Star** třída reprezentuje hvězdu. V metodě **compute()** se aktuální souřadnice počítají pomocí rovníkových souřadnic II. druhu.
- **Sun** třída reprezentující Slunce. V metodě **compute()** se aktuální pozice počítá pomocí konkrétního algoritmu (kapitola 3.2).
- **Moon** třída reprezentující Měsíc. V metodě **compute()** se aktuální pozice počítá pomocí konkrétního algoritmu (kapitola 3.3).
- **StencilBuffer** je implementací jednoduchého stencil bufferu udržujícího identifikátor objektu zakódovaného pomocí RGB v HTML5 canvasu (kapitola 5.4). Metoda **set()** přijímá jako parametry souřadnice [x,y], kde se objekt nachází a jeho id. ID objektu můžeme získat zpět pomocí metody **get()**
- **RadialBuffer** třída *dekoruje* třídu **StencilBuffer** a umožňuje uložit do bufferu souřadnice ve tvaru [x,y,poloměr].
- **GeoPositionMessanger** třída přenášející informaci o geografických souřadnicích pozorovatele.
- **JulianDate** je třídou reprezentující juliánské datum a práci s časem. Obsahuje především metody pro převod mezi systémy počítání času.

6 Výkon aplikace

Při vývoji jsem se setkal s mnoha faktory zpomalující běh aplikace. Ty jsem se snažil co nejvíce eliminovat. Zde je výčet těch nejpodstatnějších.

- Nejvíce problematickým místem je vykreslování grafických prvků do canvasu. Pro vykreslení hvězd jsem nejdříve používal generátor gradientu přímo v canvasu. Tento postup byl funkční, ale díky svému nízkému výkonu nepoužitelný. Generátor jsem nakonec nahradil předgenerovanými statickými obrázky. Výkon aplikace tímto razantně stoupl.
- Není potřeba vykreslovat objekty, které se zobrazují mimo canvas. Objekty jdou vykreslit i za hranice rozměrů canvasu, uživatel je však nevidí. Toto počítání zbytečně zvyšuje počet operací nad canvasem bez viditelných výsledků. Při zjištění, že by se objekt vykreslil mimo canvas, se nevykreslí.
- Canvas dovoluje vykreslit objekt na souřadnice udané desetinnou čárkou. Pomocí anti-aliasingu se potom snaží vyhladit hrany. Je dobré souřadnice udávat v celých číslech.
- Operace nad canvasem je dobré provádět hromadně. Pokud například potřebujeme vykreslit sadu linek, navrhneme algoritmus tak, aby se vytvořili nejlépe *jedním tahem*
- Je vhodné si předgenerovat grafické prvky, které se na plátně nemění. Pokud máme scénu, do které se jednou vykreslí sada objektů a po celý průběh aplikace scéna zůstane stejná, uložíme si ji do png obrázku nebo sekundárního canvasu. Je zbytečné objekty vykreslovat v každé další iteraci znovu.
- Pro vyšší výkon je důležité vyhnout se zbytečnému přepínání stavu/vlastností canvasu. Pokud potřebujeme vykreslit sadu objektů několika barev, vyplatí se nejdříve vykreslit všechny objekty jedné barvy, poté všechny objekty další barvy a tak dále.
- requestAnimationFrame je nová funkcionalita javascriptového API. Jako parametr přijímá funkci vykreslující objekty do canvasu a chytře vykreslování řídí podle potřeby a dostupných prostředků. Pro vykreslovací smyčku bychom mohli použít funkci setInterval(), ta však neřeší kontext prohlížeče. Provádí se stále, i když okno s aplikací není aktivní a nestará se o to, co se na obrazovce změnilo a co zůstalo stejné.
- V Dartu, pokud předem známe datový typ čísla, je vhodné použít přímo konkrétní typ, než obecný typ num. Ušetří se čas zjišťováním typu a přetypováním.
- Je dobré vyhnout se zbytečným operacím, voláním a přetypováním.

6.1 Srovnání výkonu aplikace

Zajímavou částí práce je finální srovnání výkonu aplikace, tj. kódu napsaného v Dartu a jeho zkompilovanou variantou. Propastné rozdíly ve výkonu byly také mezi prohlížeči, ve kterých se aplikace spouštěla.

Testy byly spuštěny na stroji Intel Pentium i3 2.67Ghz, 4 GB RAM, Ubuntu Linux 13.04. Testovanými prohlížeči byly Google Chrome 33.0.1750.152, Dartium 32.0.1700.58 a Firefox 26.0.

Aplikace je spuštěna ve dvou na sobě nezávislých smyčkách. Jedna smyčka se stará o výpočet souřadnic objektů, druhá o jejich vykreslení. Nebyl proto problém otestovat výkon každé z těchto komponent nezávisle. Jeden test se vždy skládal z 1000 iterací smyčky a výstupem bylo pole s časem výpočtu v milisekundách. Z něj jsem získal medián a aritmetický průměr.

6.1.1 Výkon doménové vrstvy

Test	Medián (ms)	Aritmetický průměr (ms)
1.	1	1.556
2.	1	1.518
3.	1	1.346
4.	1	1.515
5.	1	1.52

Tabulka 1: Dart, Dartium

Test	Medián (ms)	Aritmetický průměr (ms)
1.	1	1.317
2.	1	1.501
3.	1	1.663
4.	1	1.275
5.	1	1.139

Tabulka 2: JavaScript, Google Chrome

Test	Medián (ms)	Aritmetický průměr (ms)
1.	3	3.405
2.	3	3.623
3.	3	3.492
4.	3	3.431
5.	3	3.651

Tabulka 3: JavaScript, Firefox

Překvapivým výsledkem tohoto testu bylo, že zkompilovaný kód v JavaScriptu byl v Chrome rychlejší, než kód v Dartu a Dartiu. Jedním z důvodů může být starší verze Dartia oproti Google Chrome. případně ještě ne moc optimalizované komponenty jazyka nebo virtuálního stroje. Výsledek je však i přesto velmi slušný a výpočet souřadnic přibližně 1500

objektů netrval ve více než polovině případů delší dobu než 1 ms. Hůř na tom dopadl Firefox s výpočetním časem přibližně 3 ms.

6.1.2 Výkon prezentační vrstvy

Test	Medián (ms)	Aritmetický průměr (ms)
1.	31	32.622
2.	33	34.778
3.	34	35.609
4.	30	31.556
5.	32	33.158

Tabulka 4: Dart, Dartium, 1366x768

Test	Medián (ms)	Aritmetický průměr (ms)
1.	37	37.683
2.	36	36.238
3.	39	39.215
4.	33	33.203
5.	37	37.599

Tabulka 5: Dart, Dartium, 1920x1080

Test	Medián (ms)	Aritmetický průměr (ms)
1.	41	42.096
2.	43	44.21
3.	42	42.374
4.	40	41.268
5.	46	46.975

Tabulka 6: JavaScript, Google Chrome, 1366x768

Test	Medián (ms)	Aritmetický průměr (ms)
1.	44	45.262
2.	48	51.632
3.	50	52.016
4.	46	49.448
5.	50	51.64

Tabulka 7: JavaScript, Google Chrome, 1920x1080

Test	Medián (ms)	Aritmetický průměr (ms)
1.	74	76.792
2.	72	75.258
3.	81	85.19
4.	76	78.814
5.	74	77.115

Tabulka 8: JavaScript, Firefox, 1366x768

Test	Medián (ms)	Aritmetický průměr (ms)
1.	74	75.876
2.	78	82.298
3.	86	92.518
4.	77	81.518
5.	75	80.146

Tabulka 9: JavaScript, Firefox, 1900x1080

Oproti doménové vrstvě je kód prezentační vrstvy v Dartu rychlejší, než kód v zkompilovaném JavaScriptu. Jde zde také vidět, že výkon canvasu je nepřímo úměrný jeho velikosti. Větší okno znamená nižší výkon při vykreslování. Běh v Chrome/Dartiu je plynulý s přibližně 25-30 FPS; problémem je však prohlížeč Firefox, ve kterém se v extrémních případech při velkém rozlišení dostáváme na nějakých 11-12 snímků za sekundu. Při nižším rozlišení v nejhorším případě na 13-14.

V každé iteraci dochází k úplnému překreslení canvasu. Abychom získali vyšší výkon a lepší hodnotu FPS, mohli bychom aplikovat některé techniky pro jeho vylepšení, jako je překreslení pouze určité části canvasu (té, která se od posledního vykreslení změnila) nebo použít některé z výše zmíněných knihoven pro práci s grafikou s již optimalizovanými algoritmy.

7 Závěr

7.1 Možný budoucí rozvoj

Z výše popsané architektury aplikace je možné odhadnout, že není problém využít model s výpočty a použít jej v jiném projektu, případně zaměnit stávající view za jiné. Pokud bychom zůstali u 2D podoby aplikace, dala by se využít výše zmíněná knihovna StageXL. Ta se pravidelně aktualizuje a jak se vyvíjí Dart, vyvíjí se i tento balíček. Pracují na něm lidi, kteří se zajímají o počítačovou grafiku a vývoj her. Znamená to mimo jiné, že bude optimalizovaná a její nasazení bude znamenat lepší výkon aplikace.

Další z možností je využití 3D grafiky ve webovém prohlížeči – WebGL. S čím dál lepší podporou v prohlížečích a hardwarovou akcelerací by se pomocí WebGL dosáhlo kvalitnějšího vzhledu aplikace. Planety by mohly být 3D koulemi s namapovanou texturou a uživatel by si mohl planetou rotovat, zobrazovat větší detaily a pohybovat se kamerou prostorem. Inspirací pro vývoj by mohla být aplikace Stellarium nebo velmi zdařilý webový projekt **100,000 Stars** napsaný za pomocí JavaScriptu, knihovny three.js a WebGL.

Co se týče modelu a výpočtu souřadnic objektů, bylo by vhodné optimalizovat samotný výpočet a zároveň zvýšit přesnost vypočtených souřadnic. Některé objekty mají v závislosti na čase odchylku okolo 7", což je poměrně velký rozdíl oproti skutečné pozici. V některých případech by šly nahradit stávající algoritmy za jiné, přesnější; v jiných by bylo potřeba zahrnout další vlivy působící na tělesa, jako je precese, nutace a perturbace.

7.2 Závěr

Výsledkem této bakalářské práce je funkční mapa hvězdné oblohy pro webové prohlížeče. Aplikace, díky své architektuře, je připravena pro další rozvoj, kterým může být nová prezentační vrstva nebo vylepšená vrstva doménová.

Práce byla realizována v jazyku Dart, který se osvědčil pro vývoj větších webových aplikací. Výsledkem bylo také zhodnocení jeho výkonu v porovnání se zkompilovanou variantou v jazyce JavaScript.

Realizace byla přínosná hned v několika ohledech. Věřím totiž, že jazyk Dart má ve světě internetu své místo a budoucnost. Seznámil jsem se tudíž s technologií, ve které bych rád dál vyvíjel a zdokonaloval se. Bylo také zajímavé řešit problémy související s počítačovou grafikou – ať už to bylo promítání, identifikace objektů nebo snaha o lepší výkon aplikace. V neposlední řadě jsem se dozvěděl užitečné informace o základech sférické astronomie. Hvězdná obloha mě fascinovala již od dětských let a nyní mám lepší představu o tom, jak *„funguje“*.

Reference

[1] Meeus Jean: Astronomical Algorithms 2nd ed, Willman-Bell, Inc., 1998, ISBN 0-943396-61-1

[2] Karkoschka Erich: Astronomický atlas hvězdné oblohy. Vydání 1., Ostrava, 1995. ISBN 80-85606-67-4

[3] Pavel Miroslav: Planetátrium na PDA, Diplomová práce, ÚK/Sklad diplomových prací, 2003. http://hdl.handle.net/10084/45742

[4] Široký Jaromír, Široká Miroslava: Základy astronomie v příkladech. Vydání 2., Praha, Státní pedagogické nakladatelství, 1973

[5] Caglow: Planetary Positions with VSOP87 [online]. Dostupné z WWW: http://www.caglow.com/info/compute/vsop87

[6] Kalendář: Výpočet ekliptikálních souřadnic Měsíce [online]. Dostupné z WWW: http://kalendar.beda.cz/vypocet-ekliptikalnich-souradnic-mesice

[7] Nebeská mechanika: Základní pojmy [online]. Dostupné z WWW: http://nebmech.astronomy.cz/POJMY/pojmy.htm

[8] Physik und Astronomie: Astronomical Algorithms [online]. Dostupné z WWW: http://www.geoastro.de/elevaz/basics/meeus.htm

[9] Positional Astronomy: Conversion between horizontal and equatorial [online]. Dostupné z WWW: http://star-www.st-and.ac.uk/ fv/webnotes/chapter7.htm

[10] Přesný čas atomové hodiny [online]. Dostupné z WWW: http://www.presny-cas-atomove-hodiny.cz/atomove-hodiny-meri-presny-cas-unikatni-technologii

[11] Schlyter Paul, Computing planetary positions [online]. Dostupné z WWW: http://www.stjarnhimlen.se/comp/tutorial.html

[12] SkyBeautiful: Download "Hipparcos New Reduction"Data with calculated star distances [online]. Dostupné z WWW: http://www.skybeautiful.com/downloads/

[13] Weisstein, Eric W. "Spherical Coordinates."From MathWorld–A Wolfram Web Resource [online]. Dostupné z WWW: http://mathworld.wolfram.com/SphericalCoordinates.html

[14] Weisstein, Eric W. „Stereographic Projection.“ From MathWorld - A Wolfram Web Resource [online]. Dostupné z WWW: http://mathworld.wolfram.com/StereographicProjection.html

[15] Dart: Dart VM and dart2js Performance [online]. Dostupné z WWW: https://www.dartlang.org/performance/